一、内存管理机制

先看一段代码：

[复制代码](javascript:void(0);)

<?php

//内存管理机制

var\_dump(memory\_get\_usage());//获取内存方法，加上true返回实际内存，不加则返回表现内存

$a = "laruence";

var\_dump(memory\_get\_usage());

unset($a);

var\_dump(memory\_get\_usage());

//输出(在我的个人电脑上, 可能会因为系统,PHP版本,载入的扩展不同而不同):

//int 240552

//int 240720

//int 240552

[复制代码](javascript:void(0);)

定义变量之后，内存增加，清除变量之后，内存恢复（有些可能不会恢复和以前一样），好像定义变量时申请了一次内存，其实不是这样的，php会预先申请一块内存，不会每次定义变量就申请内存。

首先我们要打破一个思维: PHP不像C语言那样, 只有你显示的调用内存分配相关API才会有内存的分配. 也就是说, 在PHP中, 有很多我们看不到的内存分配过程.  
比如对于:  
**$a = "laruence";  
隐式的内存分配点就有:**1.1. 为变量名分配内存, 存入符号表  
2.2. 为变量值分配内存  
所以, 不能只看表象.  
第二, 别怀疑,PHP的unset确实会释放内存, 但这个释放不是C编程意义上的释放, 不是交回给OS.  
对于PHP来说, 它自身提供了一套和C语言对内存分配相似的内存管理API:

emalloc(size\_t size);

efree(void \*ptr);

ecalloc(size\_t nmemb, size\_t size);

erealloc(void \*ptr, size\_t size);

estrdup(const char \*s);

estrndup(const char \*s, unsigned int length);

这些API和C的API意义对应, 在PHP内部都是通过这些API来管理内存的.  
当我们调用emalloc申请内存的时候, PHP并不是简单的向OS要内存, 而是会像OS要一个大块的内存, 然后把其中的一块分配给申请者, 这样当再有逻辑来申请内存的时候, 就不再需要向OS申请内存了, 避免了频繁的系统调用.  
**比如如下的例子:**

[复制代码](javascript:void(0);)

var\_dump(memory\_get\_usage(true));//注意获取的是real\_size

$a = "laruence";

var\_dump(memory\_get\_usage(true));

unset($a);

var\_dump(memory\_get\_usage(true));

//输出

//int 262144

//int 262144

//int 262144

[复制代码](javascript:void(0);)

也就是我们在定义变量$a的时候, PHP并没有向系统申请新内存.同样的, 在我们调用efree释放内存的时候, PHP也不会把内存还给OS, 而会把这块内存, 归入自己维护的空闲内存列表. 而对于小块内存来说, 更可能的是, 把它放到内存缓存列表中去(后记, 某些版本的PHP, 比如我验证过的PHP5.2.4, 5.2.6, 5.2.8, 在调用get\_memory\_usage()的时候, 不会减去内存缓存列表中的可用内存块大小, 导致看起来, unset以后内存不变).

$a = "hello";

//定义变量时，存储两个方面：  
//1.变量名，存储在符号表

//2.变量值存储在内存空间

//3.在删除变量的时候，会将变量值存储的空间释放，而变量名所在的符号表不会减小（只增不减）

**只增不减的数组**Hashtable是PHP的核心结构, 数组也是用她来表示的, 而符号表也是一种关联数组, 对于如下代码:

[复制代码](javascript:void(0);)

var\_dump(memory\_get\_usage());

for($i=0;$i<100;$i++)

{

$a = "test".$i;

$$a = "hello";

}

var\_dump(memory\_get\_usage());

for($i=0;$i<100;$i++)

{

$a = "test".$i;

unset($$a);

}

var\_dump(memory\_get\_usage());

[复制代码](javascript:void(0);)

我们定义了100个变量, 然后又按个Unset了他们, 来看看输出:

//int 242104

//int 259768

//int 242920

Wow, 怎么少了这么多内存?  
这是因为对于Hashtable来说, 定义它的时候, 不可能一次性分配足够多的内存块, 来保存未知个数的元素, 所以PHP会在初始化的时候, 只是分配一小部分内存块给HashTable, 当不够用的时候再RESIZE扩容。而Hashtable, 只能扩容, 不会减少, 对于上面的例子, 当我们存入100个变量的时候, 符号表不够用了, 做了一次扩容, 而当我们依次unset掉这100个变量以后, 变量占用的内存是释放了(118848 – 104448), 但是符号表并没有缩小, 所以这些少的内存是被符号表本身占去了…

二、垃圾回收机制

PHP变量存储在一个zval容器里面的  
1.变量类型 2. 变量值 3. is\_ref 代表是否有地址引用 4. refcount 指向该值的变量数量

变量赋值的时候：is\_ref为false， refcount为1

$a = 1;

xdebug\_debug\_zval('a');

echo PHP\_EOL;//换行符，提高代码的源代码级可移植性

输出：

a:

*(refcount=1, is\_ref=0)*,

int

1

将变量a的值赋给变量b，变量b不会立刻去在内存中存储值，而是先指向变量a的值，一直到变量a有任何操作的时候

$b = $a;

xdebug\_debug\_zval('a');

echo PHP\_EOL;

输出：

a:

*(refcount=2, is\_ref=0)*,

int

1

$c = &$a;

xdebug\_debug\_zval('a');

echo PHP\_EOL;

xdebug\_debug\_zval('b');

echo PHP\_EOL;

输出：

a:

*(refcount=2, is\_ref=1)*,

int

1

b:

*(refcount=1, is\_ref=0)*,

int

1

因为程序又操作了变量a，所以变量b会自己申请一块内存将值放进去。  
所以变量a的zval容器中refcount会减1变为1，变量c指向a，所以refcount会加1变为2，is\_ref变为true

垃圾回收  
1.在5.2版本或之前版本，PHP会根据refcount值来判断是不是垃圾  
如果refcount值为0，PHP会当做垃圾释放掉  
这种回收机制有缺陷，对于环状引用的变量无法回收

环状引用：

$attr = array("hello");

$attr[]= &$attr;

xdebug\_debug\_zval('attr');

echo PHP\_EOL;

输出：

attr:

*(refcount=2, is\_ref=1)*,

**array** *(size=2)*

0 => *(refcount=1, is\_ref=0)*,

string

'hello' *(length=5)*

1 => *(refcount=2, is\_ref=1)*,

*&***array**

2.在5.3之后版本改进了垃圾回收机制  
如果发现一个zval容器中的refcount在增加，说明不是垃圾  
如果发现一个zval容器中的refcount在减少，如果减到了0，直接当做垃圾回收  
如果发现一个zval容器中的refcount在减少，并没有减到0，PHP会把该值放到缓冲区，当做有可能是垃圾的怀疑对象  
当缓冲区达到临界值，PHP会自动调用一个方法取遍历每一个值，如果发现是垃圾就清理

Where is a will ,there is a way!